Proposition de projet

Projet `Certilab`

Spécifications formelles, certification de logiciel

Draft

18 janvier 1999

Mots clés: spécification, sémantique, démonstration, preuve, logiciel, certification, protocole, télécommunications, Coq.

1 Composition de l'équipe

Joëlle Despeyroux (CR INRIA, responsable scientifique)
André Hirschowitz (professeur UNSA)
Guillaume Gillard (doctorant, 1997-2000)
Marino Miculan (post-doctorant, du 1er mai au 31 décembre 99)
Nathalie Balax (assistante de projet)

2 Introduction

Le thème de notre projet est la *certification de logiciel*. Les programmes sont souvent écrits dans des langages et outils spécialisés; leur certification nécessite donc deux parties: d'une part la validation d'outils généraux tels que des compilateurs, d'autre part la preuve de correction des programmes eux-mêmes. Notre projet recouvre ces deux phases de développement des logiciels.

La preuve de correction de compilateurs (pris dans leur intégralité: vérification de propriétés statiques, dont vérification ou inférence des types, puis traduction vers un formalisme intermédiaire ou vers du code) est l'exemple typique de *preuve de propriétés de langage*.

Dans le cas idéal où l'on peut extraire un programme à partir de sa spécification, la *preuve de correction de programmes* devient *programmation certifiée*. Nous nous attacherons à considerer des programmes de toute sorte, des glaneurs de cellule-mémoire (gc) aux protocoles.

Ce thème de recherche (spécification et preuves sur machine) est en plein essor. De nombreux laboratoires de recherche industriels (Intel, Microsoft, ...) sont demandeurs de chercheurs ou ingénieurs ayant une formation large dans le domaine. De plus, cette demande devrait encore grandir, vu que l'informatique est de plus en plus utilisée dans des domaines critiques pour la vie humaine (comme le nucléaire, l'industrie avionique ou ferroviaire), ou bien dans des domaines où les enjeux financiers sont importants (comme le spatial, les télécommunications ou le domaine bancaire).

Notre projet de recherche est *transversal.* Notre objectif est de pouvoir proposer notre expérience à toute équipe intéressée à formaliser des preuves en machine. Comme nous le verrons plus loin, nous avons déjà des collaborations dans ce sens avec différentes équipes, à l'INRIA comme à l'extérieur.

Notre démarche est essentiellement pragmatique, dirigée par des applications concrètes; notre projet est essentiellement *applicatif.* L'objectif est de développer des méthodes et des bibliothèques de spécifications et de preuves dans le domaine de la certification de logiciel.

Nous nous intéressons à la chaîne de développement complète: *spécification, prototypage et preuves*, réalisée en machine en utilisant des systèmes comme le système Coq [BBC$^+$97, HKPM97]. Ce système a pour nous une conjugaison unique d'avantages: ses fondements théoriques sont solides ([PM92, Wer94, Gim96, PM96],...), on peut extraire un programme à partir d'une spécification [Par95b, Par95a, Par97] et de plus le noyau du système a lui-même été certifié [Bar95, BW97].

La suite de cette proposition suit le plan du rapport d'activité annuel INRIA, auquel nous avons ajouté notre programme de travail sur 3 ans. Nous commencerons par notre programme de recherche, suivi d'une présentation des domaines d'application envisagés et de notre programme de travail sur 3 ans. Une courte section sera consacrée aux aspects logiciels. Puis les sections "valorisation industrielle" et "collaborations scientifiques" expliciteront les contrats, contacts et collaborations présents ou envisagés dans un futur proche. Enfin une dernière section décrira la composante "diffusion des résultats" de notre projet.

3 Programme de recherche

Notre recherche est essentiellement dirigée par les applications à moyen terme et à long terme. Ces applications sont multiples, et peuvent être classées en deux catégories: les preuves de propriétés de langages et les preuves de correction de programmes et programmes certifiés. Nous avons déjà acquis une certaine compétence dans le premier domaine. Nous nous proposons de compléter notre programme de recherche en abordant le second, dans lequel la demande industrielle s'annonce très forte à plus ou moins brève échance, quand elle ne l'est pas déjà.

3.1 Preuve de propriétés de langages

Participants: *Joëlle Despeyroux, André Hirschowitz, Guillaume Gillard, Marino Miculan.*

C'est un domaine dans lequel nous avons acquis une compétence reconnue depuis de nombreuses années, aussi bien sur le plan pratique (étude d'exemples, développement d'un logiciel [Des90, Des92]) que sur le plan théorique (élaboration de méthodes et proposition de systèmes de types -théories typées).

3.1.1 Étude d'exemples

Participants: *Joëlle Despeyroux, André Hirschowitz, Guillaume Gillard, Marino Miculan.*

Deux exemples typiques de preuve de propriétés de langages sont la preuve de conservation des types d'un langage lors de son exécution, et la preuve de correction d'un compilateur. La première propriété assure une certaine cohérence entre le système de vérification de type et les règles d'évaluation d'un langage. Ce type de preuve, essentiel pour les concepteurs du langage, donne par ailleurs au programmeur une certaine confiance dans les outils qu'il utilise.

Mini langages: Imp et Mini-ML

Nous avons réalisé l'étude de nombreux exemples sur de petits langages impératifs ou applicatifs classiques [CDD+85, CDDK86, DH94, DFH95, Ter95b, Gil97] sur papier en 86, en machine depuis, principalement dans le système Coq.

Ces exemples sont repris dans une présentation récente dans les notes de cours rédigées pour le Dea MDFI de Marseille [Des98].

Esterel et Lustre

Delphine Kaplan-Terrasse, ancien doctorant dans notre projet, a réalisé la certification du noyau du compilateur d'Esterel en Coq [KT96]. Cette preuve interesse plusieurs compagnies industrielles dont Dassault.

Dans le cadre du projet Génie 2, nous collaborons avec les équipes de recherche de Dassault-Aerospatiale sur la certification du compilateur du langage Lustre, langage réactif à flot de données.

Logiques modales

Marino Miculan a formalisé plusieurs logiques modales: la logique modale simple, la logique dynamique introduite par Furio Honsell et le μ-calcul de Kozen. Il s'est surtout attaché à definir les différentes "relations de conséquence" (modèles) possibles de ces logiques, mais il les a aussi formalisées dans LF et dans le système Coq. Cette démarche, appliquée également à différentes variantes de la sémantique opérationelle et à la logique de Hoare, constitue l'originalité et l'intérêt de sa thèse [Mic97].

Le π-calcul

La difficulté principale, et l'intêret de cette exemple est la description du phénomène d'extrusion de portée des variables.

Une formalisation très complète du π-calculus, comprenant la preuves de correction de différentes techniques de bi-simulation, a été réalisée dans Coq par Daniel Hirschkoff [Hir97, Hir99], au CERMICS, en utilisant les codes de Bruijn. Cette approche n'est pas la meilleure. Cependant, l'étude réalisée devrait nous fournir une excellente base de travail pour la même étude avec un meilleur choix de description des variables.

Loic Henry_Greard, dans le cadre d'un stage de Dea co-encadré par Joëlle Despeyroux et Davide Sangiorgi de l'équipe MEIJE, a réalisé [Hen98] une preuve de la propriété de conservation des types pour un π-calcul monadique en adaptant une technique due à James Mc Kinna et Randy Pollack (3.1.2). Cette approche nous semble être aujourd'hui la meilleure pour décrire un langage fonctionnel contenant des primitives inspirées du π-calcul; dans l'attente d'un système aussi puissant que Coq permettant la récursion sur des termes de types fonctionnels.

En collaboration avec Davide Sangiorgi, Joëlle Despeyroux a écrit une spécification d'un π-calculus monadique [DS99], en utilisant directement les fonctions de Coq (on appelle ce type de description une spécification d'ordre supérieur -3.1.2). Cette approche fournit la description la plus concise et la plus élégante, en particulier du phénomène d'extrusion de portée. Une preuve de conservation des types utilisant cette spécification est en cours de réalisation. Les premiers résultats montrent que la spécification, non seulement est élégante, mais aussi, et peut-être surtout, facilite les preuves.

Marino Miculan et Furio Honsell ont aussi écrit, de manière indépendante, une spécification d'ordre supérieur d'un π-calculus polyadique en Coq [DS99, HMS98]. Cette spécification est moins élégante que la précédente. Ceci dit, l'étude réalisée est beaucoup plus poussée plusqu'elle formalise une partie substantielle de la théorie d'une bisimulation (la bisimulation forte retardée de Milner).

Langages concurrents et à objets

Nous avons abordé récemment (été 1997) l'étude des langages parallèles [Hen98] et à objets [Lau97], par le biais des calculs (π-calcul de Milner et sigma-calcul de Martin Abadi et Luca Cardelli). L'objectif de ces études est de construire les briques de base qui permettront de formaliser des langages comme OCaml ou Java, par exemple, de manière raisonnable. C'est ce type de langages (les langages concurrents et à objets) qui nous intéresse le plus à court et moyen terme, dans une perspective à la fois de recherche et de contrats avec l'industrie.

L'objectif de la thèse de Guillaume Gillard, commencée en octobre 97, est de formaliser des calculs ou langages concurrents et à objets, basés sur le π-calcul et le sigma-calcul, dans le système Coq. Guillaume a étudié plusieurs propositions de tels calculs et a commencé à formaliser un calcul proposé récemment par Andrew Gordon, en utilisant la technique de représentation des langages introduite par ce même Andrew Gordon, avec Tom Melham [GM96] (3.1.2).

3.1.2 Méthodes

Participants: *Joëlle Despeyroux, André Hirschowitz, Guillaume Gillard.*

Nous avons proposé une méthode de preuve de correction de traduction en Sémantique Naturelle [Des86], qui fait toujours référence dans le domaine des preuves de correction de compilateur, plus de dix ans après.

Aussi surprenant que cela puisse paraître, il n'y a toujours pas de concensus sur le choix de la représentation des variables dans un langage fonctionnel.

Description des variables dans un langage fonctionnel

On dispose à l'heure actuelle de quatre familles de méthodes pour décrire les variables d'un langage de programmation fonctionnel, et leur manipulation, c'est à dire essentiellement la substitution, à α conversion près (un terme est défini modulo le renommage de ses variables). Nous connaissons bien trois d'entre elles, que nous avons encore expérimentées cette année dans le système Coq, sur le π-calcul [Hen98].

La technique la plus connue est celle des indices de de Bruijn. Les variables sont implémentées par des entiers, représentant leur profondeur dans le terme, i.e. le nombre de liaisons entre leur occurence et la racine du terme. Par exemple, les termes $\lambda x.x$ et $\lambda x.\lambda y.x$ seront respectivement représentés par les termes (lam 0) et (lam (lam 1)). Bien sûr, les termes sont bien définis à α-conversion près, mais on imagine aisément tous les problèmes causés par ce type de représentation. Les termes étant eux-mêmes déjà illisibles, les spécifications le sont naturellement aussi, d'autant plus qu'un tel terme doit être complètement réécrit lorsqu'il est plongé dans un autre terme (les entiers doivent tous être décallés)! Ce problème rend les preuves arbitrairement compliquées. Dans certains développements (λ et π-calculs, par exemple [Hue94, Hir97, Hir99]), les trois quarts du développement (spécification et preuves) concernent la manipulation des codes de Bruijn, et non la sémantique elle-même. La preuve de la propriété de conservation des types (simples) pour un π-calcul polyadique semble déjà déraisonnable, sinon impossible, dans cette approche. Cette technique, encore très utilisée, paraît donc hautement inadaptée à la formalisation des langages de programmation. Cependant, il existe un cas où l'utilisation des codes de de Bruijn est tout à fait indiquée: le cas du *boot-strap* d'un langage implémenté en utilisant ces codes. C'est le cas de la formalisation de Coq en Coq, réalisée par Bruno Barras [Bar95, BW97].

Nous explorons en ce moment une technique due à Andrew Gordon et Tom Melham [GM96]. Cette méthode, implémentée dans le système HOL, consiste à spécifier les termes du lambda-calcul à α-conversion près, pour ensuite définir les termes du langage choisi en utilisant ces termes pour formaliser les liaisons. À notre connaissance, cette méthode n'a encore jamais été vraiment utilisée.

James McKinna et Robert Pollack utilisent pour spécifier des règles de typage (typage des théories typées) une technique d'ordre un intéressante [MP93, MP99, Pol95] qui distingue les variables libres (formalisées par des *paramètres*) des variables liées (formalisées par des *variables*). Les descriptions sémantiques renomment judicieusement les variables d'un terme en paramètres avant tout traitement du terme, pour effectuer la substitution inverse à la fin du traitement. Ainsi, il ne peut pas y avoir de phénomène de "capture de variable" lors d'une substitution. (L'introduction d'une variable déjà liée dans un terme nécessite normalement son renommage préalable pour éviter toute capture.) James McKinna et Robert Pollack ont ainsi formalisé LEGO en LEGO (LEGO [LP92, Pol94] et COQ sont des systèmes très voisins). Leur approche nous semble être aujourd'hui la meilleure pour décrire un langage fonctionnel, ou dont le noyau est fonctionnel, contenant ou non des primitives inspirées du π-calcul [Hen98], dans l'attente d'un système aussi puissant que COQ permettant la récursion sur des termes de types fonctionnels.

Enfin, justement, il existe une méthode qui permet d'utiliser directement les fonctions du système choisi (COQ par exemple) pour formaliser les notions de variables liées et de substitution du langage à décrire. On appelle ce type de description une spécification d'ordre supérieur. L'idée de base de la méthode est de représenter les variables du langage spécifié (le langage dit "objet") par les variables du langage utilisé pour spécifier (le langage dit "méta"). Ainsi, un constructeur d'un langage L sera défini par des déclarations du type: $lam : (L \Rightarrow L) \Rightarrow L$. Le fils d'un noeud lam est ici une fonction (méta).

Cette technique permet également de formaliser les *side conditions* (variable libre dans une hypothèse) et les changements de contexte (déchargement d'une hypothèse en déduction naturelle) dans les règles d'inférence. Elle fournit souvent la description la plus concise et la plus élégante d'un langage. De plus, par voie de conséquence, les preuves sont aussi grandement facilitées. L'utilisateur peut se concentrer sur les difficultés de son langage, sans devoir ré-implémenter les fonctions, que le système a déjà implémentées pour lui.

Par ailleurs, la définition d'un type inductif fournit des schémas de récurrence et des principes d'induction qui permettent de faire des preuves générales sur des données de ce type. En particulier, si l'on représente la syntaxe d'un langage par un type inductif, on peut utiliser les principes associés pour faire des preuves sur la sémantique de ce langage.

Le problème est qu'une syntaxe abstraite d'ordre supérieur, définie de manière naïve, ne forme pas un type inductif. En collaboration avec d'autres chercheurs (notamment Frank Pfenning à CMU), nous travaillons depuis 1992 sur plusieurs solutions à ce problème, qui était ouvert depuis 1987. La solution à long terme passe par la conception d'une nouvelle théorie typée (voir section 3.1.3). La solution à court terme consiste à inventer de nouvelles méthodes de description des langages dans les systèmes existants. Nous avons proposé deux solutions pour cela, dans le système COQ.

En collaboration avec Amy Felty (ATT Bell Labs), nous avons défini une nouvelle notion: la *syntaxe abstraite d'ordre supérieur restreint* [DFH95] qui permet d'utiliser à la fois la syntaxe d'ordre supérieur et les types inductifs. Cette approche est basée sur une idée très simple: définir les fonctions du langage sur un ensemble de variables prédéfini, ce qui conduit à des définitions de constructeurs d'un langage L du type: $lam : (var \Rightarrow L) \Rightarrow L$. La syntaxe étant trop permissive, il est nécessaire de définir un prédicat décrivant les termes "légaux" sur lesquels on travaille. Il faut aussi spécifier une opération de substitution pour chaque langage étudié. Mais cette substitution est très facile à définir et peut utiliser l'application ("méta") du système utilisé.

Dans la seconde approche, appelée *sémantique d'ordre supérieur* [DH94], l'utilisateur n'a pas besoin de décrire une opération de substitution; il peut utiliser directement l'application du système choisi. L'idée maîtresse de cette méthode, qui sera reprise dans la définition d'une nouvelle théorie typée (3.1.3), est de considérer un terme ouvert, dépendant d'une liste de variables, comme une fonction de cette liste de variables. Un prédicat, nommé *valide*, différent du prédicat utilisé dans l'approche précédente, restreint l'ensemble des termes décrits aux arbres de syntaxe abstraite. Les sémantiques sont données sur les termes valides, qui sont donc des termes fonctionnels. D'où le nom de sémantique d'ordre supérieur pour cette méthode qui prolonge à la sémantique l'idée de la syntaxe abstraite d'ordre supérieur.

Pour conclure sur ces différentes méthodes de description des liaisons dans les langages de programmation, une seule méthode ne nous intéresse pas du tout: les indices de de Bruijn. La technique introduite par Andrew Gordon et Tom Melham est la seule sur laquelle nous ne pouvons pas encore nous prononcer: nous sommes en train de l'étudier. Comme nous l'avons dit plus haut, la méthode due à James Mc Kinna et Randy Pollack nous semble aujourd'hui la meilleure pour décrire un langage fonctionnel en l'absence d'un système permettant l'utilisation conjointe de la syntaxe abstraite d'ordre supérieur et de l'induction. La technique de spécification d'ordre supérieur est la plus abstraite, la plus élégante; c'est celle que nous préfèrerons toujours chaque fois que ce sera possible; la version restreinte de cette méthode est compatible avec l'induction. Cette technique ne nous semble pas encore avoir fait ses preuves pour les langages impératifs, bien qu'il existe plusieurs expériences dans ces langages. Cependant la recherche est encore active sur ce sujet, que nous avons inclu dans nos objectifs de recherche à court terme. La notion de sémantique d'ordre supérieur que nous avons proposée représente toujours une possibilité intéressante d'utilisation de la syntaxe abstraite d'ordre supérieur dans le système Coq actuel, peut-être la plus intéressante parmi les trois ou quatre méthodes proposées à ce jour. Un excellent défi pour cette méthode semble être la preuve du théorème de Church-Rosser pour le lambda-calcul simple. Nous avons commencé à implémenter cette preuve dans Coq.

Logical Frameworks et systèmes

En ce qui concerne les *Logical Frameworks* et les systèmes, nous pensons utiliser principalement le système Coq, mais pas seulement. Il existe beaucoup d'autres systèmes: Alf, PVS, Twelf, HOL, Isabelle, etc... Nous suivrons attentivement l'évolution du système Alf, très proche de Coq. PVS est un concurrent américain sérieux de Coq; il serait surement utile d'acquérir un minimum d'expérience dans ce système. Twelf nous intéresse plus particulièrement parce qu'il est le système actuellement le mieux placé pour l'utilisation conjointe de la syntaxe abstraite d'ordre supérieur et de l'induction. HOL est plus connu (et intensivement utilisé) dans le domaine du *hardware* que nous ne comptons pas aborder pour l'instant. Isabelle a beaucoup d'atouts, et en particuler plus d'automatisme (réécriture utilisant l'unification d'ordre supérieur) que Coq, mais ce système n'a pas le langage et les fondements puissants de Coq. Plusieurs approches pour incorporer la réécriture dans Coq sont actuellement en cours de développement.

3.1.3 Théories typées

Participant: *Joëlle Despeyroux.*

Dans un premier pas vers la définition d'un système de type qui incorpore la méthodologie du système LF ("Logical Framework" développé à Edimbourg) dans des systèmes comme Coq ou Alf [ML80, ML85, Nor93, MN94] et en collaboration avec Frank Pfenning et Carsten Schürmann (Carnegie Mellon University), nous avons proposé récemment [DPS96, DPS97] un système préservant l'adéquation des représentations utilisant la syntaxe abstraite d'ordre supérieur. Ce système est un λ-calcul modal étendu avec des opérateurs de raisonnement par cas et d'itération, permettant la récursion sur des termes de type fonctionnel.

Dans son travail de thèse [Lel98b], Pierre Leleu a proposé une variante de ce système, meilleure sur plusieurs points. D'une part, il a changé le noyau modal [Lel98a, DL99a] en reprenant celui de Frank Pfenning et Hao-Chi Wong (1995), plus agréable pour l'utilisateur. D'autre part, il a remplacé les règles d'évaluation du système initial par des règles de réduction, auxquelles il a ajouté l'êta-expansion. Le système a toutes les propriétés souhaitées: préservation du typage par la réduction, confluence, normalisation forte de la réduction, et extension conservative par rapport au λ-calcul simplement typé [Lel97, DL98, DL99c]. Pierre Leleu a aussi proposé une extension partielle de son système aux produits dépendants; dans une présentation plus proche de celle du Calcul des Constructions Inductives, sans types mutuellement récursifs et avec seulement l'élimination non-dépendante pour le moment [DL99b, DL99b].

Ce sujet de recherche très difficile est très prometteur. Malheureusement, nous manquons de moyen en hommes pour continuer à y travailler, du moins pour l'instant. Nous sommes ravis de voir de nouvelles contributions apparaître dans des conférences prestigieuses comme LICS'99. En particulier, le travail de Martin Hofmann, réalisé au niveau des catégories, devrait permettre d'étendre notre système à un système généralisant le Calcul des Constructions Inductives.

3.2 Preuve de correction de programmes

Participants: *Joëlle Despeyroux, André Hirschowitz, Marino Miculan.*

C'est le domaine que nous nous proposons d'aborder. La demande industrielle est ici très forte, dans différents secteurs plus ou moins sensibles.

Dans un premier temps, afin d'acquérir une connaissance du domaine, nous commencerons par étudier plusieurs exemples, dans des domaines différents. Dans un second temps, fort de cette expérience, nous espérons pouvoir dégager de nouvelles méthodes ou/et améliorer l'existant. Tout ceci se fera dans la mesure du possible dans le cadre de stages de Dea, puis de thèses, eventuellement co-encadrés.

Le domaine est nouveau pour nous. Cela dit, ce domaine est connexe au précédent, et de plus, nous nous proposons de l'aborder avec le même outil que précédemment: le système Coq.

3.2.1 Étude d'exemples

Participants: *Joëlle Despeyroux, André Hirschowitz, Marino Miculan.*

La preuve de correction de *programmes impératifs* est théoriquement faisable dans le système Coq depuis que Jean-Christophe Filliâtre a montré comment interpréter des preuves en logique de Hoare dans ce système [Fil97]. Nous devrons expérimenter cette nouvelle possibilité.

Le *glaneur de cellule-mémoire (gc)* est l'un des outils liés à un langage. La preuve de sa correction est cruciale, et peut s'avérer très difficile, particulièrement dans le cas d'une architecture distribuée [DG94].

Le *cache* est une partie importante du système d'exploitation d'une machine. La preuve de sa correction est aussi cruciale, particulièrement avec l'avènement du *co-design* (conception en parallèle du hardware et d'une partie du software).

Plusieurs équipes de recherche, dans le monde académique comme dans l'industrie, s'intéressent au système Coq pour formaliser des preuves de correction de *protocoles de communication*. Notons en particulier l'équipe Dyade de Bull (action VIP), autour de Dominique Bolignano [Bol96] et le CNET, dans le cadre d'une collaboration avec les projets COQ et CROAP.

Notons que la plupart des preuves de correction de programmes se font sur une spécification formelle abstraite (une abstraction) de ce programme et non sur le programme lui-même. C'est notamment le cas pour les preuves de correction de protocoles.

3.2.2 Méthodes

Participants: *Joëlle Despeyroux, Marino Miculan.*

Le plus souvent, on ne prouve pas la correction du programme lui-même, mais d'une description formelle de ce programme dans différents formalismes (logique temporelle, Unity, etc). La difficulté principale étant alors de trouver le bon niveau d'abstraction de la description, tout en s'assurant (informellement) que les hypothèses faites sont réalisables! La seule approche qui ne suit pas ce schéma est l'extraction de programme.

Extraction de programme

L'extraction de programme dans le système Coq consiste en fait à développer en même temps un programme et sa spécification – qui dans un système basé sur la théorie des types, comme Coq, se trouve être justement une preuve de sa correction. Nous fondons beaucoup d'espoir sur cette technique, encore en développement, qui est l'une des originalité et l'une des grandes forces du système Coq.

Unity/TLA

Les preuves de correction de protocoles de communication sont souvent réalisées dans un système appelé Unity. Or le langage Unity a été récemment partiellement décrit sous le système Coq par Pierre Crégut et Barbara Heyd [CH97, CH96, Hey97]. Il est donc possible d'aborder les preuves de correction de protocoles dans Coq en utilisant Unity. Nous sommes très curieux d'essayer cette approche.

La logique temporelle (TLA) est beaucoup utilisée pour formaliser les preuves de correction de protocoles, entre autres domaines. Damien Doligez et Georges Gonthier l'ont notamment utilisée pour formaliser la preuve de correction d'un gc pour une architecture multi-processeurs [DG94]. C'est une approche que nous devrons étudier, et essayer par nous-mêmes.

La méthode B

La méthode B développée par J-R. Abrial dans les années 80, et toujours en développement, est encore l'une des méthodes les plus utilisées dans le monde industriel. Elle a notamment été utilisée, avec succès, pour spécifier le métro parisien. Cette méthode n'est pas aussi formelle que les autres approches dont nous avons parlé. Cependant, il existe plusieurs formalisations de cette méthode, dans différents systèmes, avec différents buts: prouver des propriétés méta-théoriques de la méthode, ou permettre la preuve des obligations de preuves générées par le système. L'une des plus récentes propositions, réalisée pour les systèmes Coq et PVS, permet les deux [BFM98]. C'est une proposition a priori très intéressante, que nous espérons pouvoir étudier. Cela dit, cette étude ne fait a priori pas partie de nos objectifs à court terme.

Proof-carrying code

Cette technique, présentée à la conférence POPL en 97, permet à un site hôte de vérifier qu'un code produit par un client par hypothèse non fiable vérifie certaines propriétés. Les applications sont nombreuses, et comprennent typiquement le code mobile, mais aussi, par exemple, les compilateurs d'un langage (fiable) de haut niveau pouvant importer du code (non fiable) d'un langage de bas niveau.

L'idée clé est que l'implémenteur du code doit fournir une preuve attestant le fait que le code respecte une certaine politique de securité definie par le site hôte. La preuve de correction doit pouvoir être réalisée avec un outil simple et efficace.

L'idée est excellente et est actuellement intensivement utilisée pour le compilateur de Standard ML dans l'équipe de Peter Lee a CMU. Il nous faudra l'expérimenter par nous-mêmes.

Interprétation abstraite

Cette méthode n'est pas une technique de preuve, mais une technique d'analyse de programme, qui permet de trouver des *bugs*, mais est surtout utilisée pour optimiser du code, typiquement un compilateur. L'analyse porte sur les variables d'un programme: leur point de définition, leurs points d'appel, et les conditions aux bords des variables (dépassement des bornes d'un tableau par exemple). C'est une technique très utilisée que nous devrons apprendre à maîtriser.

3.3 La part de rêve, ou veille technologique: mobilité, réseaux

Participant: *Joëlle Despeyroux.*

Cette section est hautement prospective... Mais n'est-ce pas là la vocation première de la recherche?

Une famille de langages de programmation est en train de voir le jour, pour décrire deux phénomènes liés aux réseaux: d'une part la mobilité (processus mobiles), d'autre part la description du réseau lui-même (notion d'ambients de Luca Cardelli). Pour l'instant, il ne s'agit pour nous que de suivre l'évolution de ces recherches: ces notions sont en train de naître; on ne dispose jusqu'ici que de différentes propositions de modèles, au mieux de calculs, rarement de langages. Toutes les propositions de calcul mobile sont plus ou moins basées sur le π-calcul (join-calcul, calcul bleu, ...) et allient avec plus ou moins de bonheur les notions de concurrence et d'objets. Elles sont relativement difficiles à comparer. La phase de formalisation et de preuve (particulièrement de preuve de correction de programmes – protocoles) n'est donc pas entamée. Cela dit, il existe déjà une communauté internationale très active sur ce sujet: projet ESPRIT CONFER comprenant notamment les projets Meije et Para à l'INRIA, Benjamin Pierce à l'université de Penn et le laboratoire de recherche de Microsoft avec Luca Cardelli et Andrew Gordon. Ce domaine de recherche est probablement l'un des domaines les plus novateurs, et prometteurs, du moment. De plus, le besoin industriel est évident. Il se pourrait que ce qui semble hautement prospectif aujourd'hui soit déjà une réalité demain, c'est à dire dans 2 ou 3 ans...

En tous les cas, ces langages nous offrent l'une des applications les plus intéressantes à plus d'un titre: d'une part ils représentent des défis certains pour nous (nous avons déjà commencé à formaliser le π-calcul), d'autre part ils nous paraissent très prometteurs. D'où une motivation double.

4 Domaines d'application

L'informatique intervient maintenant dans de nombreux domaines, souvent de manière critique, typiquement dans les secteurs du nucléaire, de l'avionique ou de l'aéronautique, ou plus récemment dans le secteur des télécommunications. Que ce soit pour des raisons de protection de vies humaines ou pour des raisons économiques, produire des *logiciels sûrs* est un enjeu important. Il existe pour ce faire deux classes de méthodes: certifier ou tester. La *certification* est la seule méthode sûre. De plus, avant de prouver, il faut *spécifier*, et de nombreuses erreurs sont souvent détectées dans cette première phase.

4.1 Conception et prototypage de langages de programmation

C'était l'objectif de la chaîne de développement CENTAUR-COQ, qui a été interrompue à la fermeture du projet CROAP. À partir d'une spécification en Sémantique Naturelle, le système CENTAUR génère automatiquement deux données: d'une part un programme Prolog qui permet d'exécuter, et donc de tester/prototyper cette spécification, et d'autre part des spécifications pour le système Coq [CT95, Ter95a, Ter95b] dans lequel les preuves peuvent être faites. Ceci fournit donc une chaîne complète: *spécification, exécution/prototypage, traduction en* Coq *et preuves*.

La partie “spécification et exécution” permet de spécifier un langage dès la phase de conception, ce qui est maintenant le cas pour tout langage moderne, depuis le langage Ada, sinon avant. Cette démarche est particulièrement intéressante dans deux cas: le cas d'un langage spécifique à un domaine (*domain specific language*), et le cas d'un langage, ou calcul, objet et produit d'une recherche de pointe (comme le calcul bleu à l'INRIA-Sophia, par exemple).

La partie “prototypage de spécifications” est particulièrement importante car elle permet de détecter de nombreuses erreurs dans les spécifications à un coût bien inférieur au coût de la preuve elle-même.

Il est certainement possible d'utiliser l'expérience acquise chez CROAP pour implémenter toute la chaîne de développement à l'intérieur de chaque système d'assistant à la preuve. En particilier, ceci ne devrait pas être trop difficile pour le système Coq, et constitue d'ailleurs l'une des priorités du projet COQ.

4.2 Logiciel embarqué

Il devient de plus en plus crucial de certifier les systèmes critiques embarqués, particulièrement dans les domaines critiques pour la vie humaine, comme le nucléaire, l'industrie avionique ou ferroviaire. Ces domaines font appel à la recherche depuis de nombreuses années. Notons par exemple que le langage Esterel est étudié comme formalisme de programmation dans plusieurs compagnies, au rang desquelles Dassault-Aviation. De son côté, Lustre, un langage réactif plus simple qu'Esterel, est effectivement utilisé dans l'industrie pour la conception de logiciel de contrôle de centrales nucléaires (Schneider-Electric), de logiciel de commande de vol (Aerospatiale (Sao+)) et de logiciel de contrôle ferroviaire (CS-transport). Dans le cadre du projet Génie 2, les équipes CROAP et MEIJE de l'INRIA collaborent avec les équipes de recherche de Dassault-Aerospatiale sur la certification des compilateurs de ces deux langages (Esterel et Lustre).

4.3 Aéronautique

L'informatique est aussi de plus en plus utilisée dans des domaines où les enjeux financiers sont importants. L'un de ces domaines est l'aéronautique, comme on l'a vu récemment avec le crash d'Ariane 5, qui aurait pu être très facilement évité. La technique utilisée pour l'instant pour trouver les *bugs* n'est pas une technique de preuve, mais simplement une technique d'analyse de programme: l'*interprétation abstraite*. Cette analyse permet typiquement de connaître les conditions aux bords des variables (dépassement des bornes d'un tableau par exemple). Une analyse relativement simple des programmes permet aussi de montrer les dépendances entre les variables, leur point de définition et leurs points d'appel. Ces techniques ont permis d'identifier de nombreux *bugs* dans le logiciel d'Ariane 5. Un des chercheurs INRIA impliqués, à l'origine du programme d'analyse utilisé, est d'ailleurs en train de créer une entreprise sur ce thème.

4.4 Télécommunications

Les télécommunications et le domaine bancaire sont aussi des domaines où les enjeux financiers sont importants. Dans ce secteur, les preuves recherchées sont des preuves de correction de protocoles ou plus généralement des preuves de sécurité. Les protocoles dont on peut prouver la correction avec nos outils de preuve interactifs ne sont encore que des protocoles jouets, bien loin des protocoles utilisés dans le monde réel. Là encore, c'est le rôle de la recherche de préparer le futur, dans ce cas pas très lointain semble-t-il.

4.5 Programmation du Web

Cette section est relativement prospective. Nous avons mentionné l'émergence de calculs et de langages permettant de décrire du code mobile, ou même un réseau. Luca Cardelli a proposé une notion d'ambients pour décrire les objets d'un réseau. Actuellement, il est très difficile de programmer un objet d'un réseau, et encore plus difficile de prouver, ou même seulement d'exprimer ses propriétés minimales de correction. On manque d'un langage dédié à cette tâche. Concevoir un tel langage est l'objectif de ce domaine de recherche très actif, qui répond à une demande évidente du secteur applicatif.

Le problème des preuves de propriétés de ces langages n'apparaîtra clairement que lorsque leur définition aura un peu avancé; le problème des preuves de correction de programmes écrits dans ces nouveaux langages intervenant dans une phase ultérieure.

5 Programme de travail sur 3 ans

Nos objectifs forment un tout cohérent, mais très large. Il nous faudra faire des choix, plus ou moins douloureux, selon le capital humain dont disposera le projet.

Nous nous fixons les objectifs minimaux suivants:

- continuer à développer des bibliothèques de spécifications et de preuves
  (Joëlle Despeyroux, André Hirschowitz, Guillaume Gillard, Marino Miculan),
- rechercher le moyen d'utiliser la syntaxe abstraite d'ordre supérieur pour décrire les langages impératifs, en particulier les langages construits sur un noyau applicatif comme ML, Caml ou OCaml
  (Joëlle Despeyroux, André Hirschowitz),
- acquérir une compétence dans le domaine de la preuve de correction de programmes, en particulier de protocoles. Les méthodes de preuves que nous pensons essayer en priorité sont l'extraction de programme, Unity et le *proof-carrying code*
  (Joëlle Despeyroux, André Hirschowitz, Marino Miculan).

6 Logiciels

Nous ne nous proposons de développer aucun logiciel. En revanche, notre activité principale sera de contribuer au développement de différents logiciels, dont deux développés à l'INRIA. D'une part nous serons, au moins partiellement, un projet applicatif du projet COQ, puisqu'une partie non négligeable de notre activité consistera en l'élaboration d'une bibliothèque pour le système Coq, développé à l'INRIA-Rocquencourt. D'autre part, nos expériences fourniront un laboratoire privilégié d'exemples pour le système CtCoq, développé à l'INRIA-Sophia.

7 Valorisation industrielle

7.1 Contrats industriels actuels

7.1.1 Dassault-Aerospatiale, Vérilog

Dans le cadre du projet Génie 2, nous collaborons avec les équipes de recherche de Dassault-Aerospatiale sur la certification du compilateur du langage Lustre, langage réactif à flot de données. La compagnie Vérilog intervient comme consultant dans cette action en fournissant gracieusement le code de son compilateur Lustre.

Delphine Kaplan-Terrasse, ancien doctorant dans notre équipe, a initialisé la preuve de correction du compilateur Esterel lors d'un séjour post-doctoral dans l'équipe MEIJE. Cette preuve, qui interesse plusieurs compagnies dont Dassault, devrait être maintenant poursuivie dans le cadre du projet Génie 2, par un doctorant qui s'est initié aux preuves de propriétés de langages dans Coq dans notre équipe (Loïc Henry-Gréard).

Les équipes de recherche de Dassault-Aérospatiale sont intéressées à collaborer avec nous sur le thème de la certification de logiciel, sur le long terme. Nous rechercherons donc à continuer de matérialiser cette collaboration dans différents contrats prenant la suite des contrats Génie, sous une autre forme.

7.2 Prospective

7.2.1 CNET

Nous avons mentionné l'interêt que le CNET porte au système Coq pour formaliser les preuves de correction de protocoles de communication. Le CNET collabore déjà avec les équipes COQ et CROAP, ainsi qu'avec l'équipe MEIJE. Il a en particulier financé une bourse de thèse sur l'élaboration d'outils d'aide au développement de grosses preuves dans le système Coq, dans l'équipe CROAP. C'est un laboratoire potentiellement interessé à collaborer avec nous sur le thème de la correction de programmes.

7.2.2 Dyade, action VIP

L'action VIP de Dyade consacre son énergie à la preuve de correction de protocoles.

Certains membres de l'équipe, notamment Jean Goubault, travaillent au niveau théorique dans le domaine de la théorie des types. Jean Goubault a en particulier travaillé sur des sujets proches des nôtres, comme l'étude méta-théorique de différentes présentations de la logique modale.

7.2.3 Harlequin

L'équipe d'Edimbourg, dans le projet Types, est en relation avec la compagnie Harlequin qui a utilisé le système Lego pour implémenter une preuve de correction de gc [Jac98]. Cette compagnie devrait être, au même titre que la précédente, un bon candidat comme partenaire industriel d'un projet ESPRIT commun avec le site d'Edimbourg.

7.2.4 Prover

L'équipe de Goteborg, dans le projet Types, est en relation avec la compagnie suédoise Prover qui utilise des méthodes formelles pour réaliser des preuves de correction de logiciel. Cette compagnie devrait être un bon candidat comme partenaire industriel d'un projet ESPRIT commun avec le site de Goteborg.

7.2.5 RATP, Digilog

La RATP utilise la méthode B depuis plusieurs années pour spécifier le métro parisien. Cette société pourrait s'interesser maintenant à Coq puisque, comme nous l'avons vu plus haut, la méthode B a été récemment formalisée dans ce système.

La méthode B est implémentée dans un logiciel commercialisé par la société Digilog Stéria, à Aix-en-provence. Cette société devrait également être interessée par l'apport du système Coq.

7.2.6 Renault

Un ancien doctorant de l'équipe COQ, Samuel Boutin, est actuellement membre de l'équipe de recherche de la société Renault, qui utilise depuis plusieurs années les méthodes formelles pour vérifier des composants informatiques de ses voitures. On peut espérer que Renault s'interesse dorénavant au système Coq pour réaliser les preuves de correction de logiciels dont elle a besoin. En particulier, Renault a besoin d'une formalisation de *statemate*, dont il existe plusieurs descriptions à comprendre.

8 Collaborations scientifiques

8.1 Positionnement par rapport aux projets INRIA

8.1.1 Projet COQ à Rocquencourt

Une partie de notre activité consistait en l'élaboration d'une nouvelle théorie des types contenant le Calcul des Constrctions Inductives. Si nous pouvons continuer cette activité, ce sera sans doute en étroite collaboration avec le projet COQ, d'autant plus que nous abordons maintenant le domaine difficile de la récursion sur des termes de type dépendant, domaine qui requiert une très haute expertise!

Ce n'est pas la volonté du groupe de traiter des applications. Cependant, afin de *boot-straper* le système, Bruno Barras a formalisé Coq en Coq [Bar95, BW97], en utilisant les codes de de Bruijn.

Nous serons, au moins partiellement, un projet applicatif du projet COQ, puisqu'une partie non négligeable de notre activité consistera en l'élaboration d'une biliothèque pour le système Coq. Ce faisant, nous contriburons sans doute à l'identification de faiblesses au niveau de la théorie ou du système, et serons demandeurs de nouvelles fonctionalités.

8.1.2 Projet CRISTAL à Rocquencourt

Depuis 1985 sur papier [CDD+85, CDDK86, Des86], puis 1992 en machine [Ter95b, Gil97, Des98], nous avons proposé et étudié de nombreuses formalisations de Mini-ML, coeur du langage Caml.

Les membres du projet CRISTAL sont toujours intéressés à formaliser le langage OCaml et, à terme, à réaliser un maximum de preuves de propriétés de ce langage. Nous avions d'ailleurs proposé une action de recherche concertée (anciennement actions incitatives) en ce sens avec le projet COQ à l'automne 1997.

Plus généralement, les membres de cette équipe participent activement à l'effort international de recherche de systèmes de types pour les langages parallèles et à objets, effort que nous suivons avec le plus grand intérêt, toujours prêts à formaliser ce qu'il apparaîtrait intéressant de formaliser en machine dans ce domaine.

8.1.3 Action LEMME à Sophia

Un cas particulier des preuves de correction de programmes est le cas des programmes certifiés, qui comprend en particulier l'ensemble des algorithmes (fonctions mathématiques) certifiés. L'action LEMME utilisant comme nous le système Coq, nos deux équipes gagneront beaucoup à partager leur experience. Cependant, a priori, les bibliothèques développées devraient être disjointes. D'autre part, malgré certains besoins communs (sous-typage, types quotients) une grande partie des problèmes rencontrés sera de nature relativement différente (extraction dans Types pour les algorithmes, dans Prop pour les programmes).

Par ailleurs, nos expériences fourniront un laboratoire privilégié d'exemples pour le système CtCoq, interface interactive pour Coq que l'action LEMME se propose de continuer de développer, sous le système Centaur, en cours de réécriture en Java.

8.1.4 Projet MEIJE à Sophia

Ce projet est constitué de deux parties relativement disjointes.

La thématique d'une partie du projet est centrée sur la notion de mobilité, avec comme objectif à long terme la définition d'un langage de programmation implémentant la notion de processus mobile. Là encore, l'une des difficultés majeures est de trouver le bon langage de type, et là encore, pour les raisons mentionnées plus haut, nous suivons l'effort international de recherche de ces langages avec le plus grand intérêt, toujours prêts à apporter notre contribution. Nous avons d'ailleurs déjà commencé, en proposant différentes formalisations du π-calcul dans le système Coq.

Le reste du projet a lui-même deux thématiques: d'une part le travail autour du langage Esterel, d'autre part le développement de méthodes et outils de preuves automatiques par vérification exhaustive de modèles finis (*model-checking*), qui s'appliquent particulièrement bien aux programmes Esterel. Le concepteur et les utilisateurs industriels du langage Esterel sont très intéressés à l'existence d'un "Esterel certifié", i.e. d'outils de compilation certifiés pour Esterel. Delphine Kaplan-Terrasse a déjà prouvé la correction du compilateur d'un noyau significatif d'Esterel [KT96]. Loïc Henry-Gréard a entamé récemment (septembre 1998) l'extension de la preuve au langage complet.

Les méthodes de démonstration automatiques (réécriture ou *model-checking*) sont orthogonales, et complémentaires, de nos méthodes interactives. Les méthodes interactives sont nettement plus puissantes, mais elles sont déraisonnables là où les méthodes automatiques sont applicables. Partant de cette évidence, de nombreux chercheurs ont proposé différentes manières de combiner ces deux approches. Si l'intégration de la réécriture dans les outils interactifs est toujours un sujet actif, l'intégration du *model-checking* semble passée de mode. Il serait intéressant de savoir pourquoi, et surtout si ce constat est définitif.

8.1.5 Action OASIS à Sophia

Cette action a pour thème la formalisation et le développement d'environnements de programmation pour les langages parallèles et à objets (type C++//, Eiffel// et Java) et les problèmes de sécurité dans ces langages (Java et Java-card). En ce qui concerne les applications, nos thématiques sont proches, exepté que nous nous intéressons davantage aux calculs et langages en développement dans la recherche qu'aux langages actuellement utilisés dans l'industrie de manière intensive. Cela dit, il sera certainement intéressant pour nous de suivre l'évolution de ce projet, et le cas échéant d'entamer une collaboration avec lui; il sera beaucoup plus proche que nous des besoins des industriels à court terme, au moins en ce qui concerne les applications Java et Java-card.

8.2 Collaborations nationales

8.2.1 ENS Lyon

Jean Duprat, membre du projet ESPRIT Types (8.3.1), a developé une preuve de correction pour un gc distribué. Son expérience nous sera certainement utile.

8.2.2 CERMICS, Paris

Nous avons suivi de près le travail de thèse de Daniel Hirschkoff au CERMICS, sous la direction de Réné Lalement, sur la formalisation du π-calcul en Coq [Hir97, Hir99]. Joëlle Despeyroux fait d'ailleurs partie du jury de thèse de Daniel Hirschkoff, dont la soutenance est prévue pour janvier 99.

8.2.3 CNAM, Paris

Catherine Dubois, dans l'équipe de Véronique Donzeau-Gouge-Viguier au CNAM à Paris, travaille sur la formalisation de Mini-ML et les preuves de correction de l'algorithme de typage W pour ce langage depuis plusieurs années. Là aussi, bien sûr, nous restons en contact étroit.

8.2.4 Appel à idées CNRS/DRET/MENESR

En novembre 1996, nous avons répondu à l'appel à idées lancé conjointement par le CNRS, la DRET et le ministère (MENESR), dans le cadre d'une action commune civil/défense: "Maîtrise des systèmes complexes réactifs et sûrs". Notre proposition, intitulée "certification de logiciel pour des langages réactifs", impliquait les équipes INRIA COQ, CROAP, MEIJE et PROTHEO. Cette proposition n'a pas eu de suite concréte.

8.2.5 Action de Recherche Concertée CRISTAL/CROAP/COQ

En avril 1997, nous avons rédigé une proposition d'action de recherche concertée (anciennement actions incitatives), impliquant les équipes INRIA CRISTAL, CROAP et COQ. Le sujet était: "preuves de propriétés du langage Objective Caml". Cette proposition a été jugée très intéressante, mais néamoins refusée par la direction de l'INRIA. Les équipes impliquées sont toujours intéressées à collaborer dans cette recherche. Une aide de l'INRIA, finançant partiellement des bourses de thèse ou des séjours post-doctoraux, serait toujours la bienvenue.

8.3 Collaborations européennes

8.3.1 *ESPRIT Working Group Types*

Depuis 1989 (conjointement avec Gilles Kahn jusqu'en juin 94 - *site leader* depuis), Joëlle Despeyroux a représenté le groupe Croap dans les projets Esprit *Basic Research Action*: *Logical Frameworks* (89-91) et "types pour les preuves et les programmmes" (92-95). Elle le représente encore dans le *Working Group Types*, depuis 96. La page web du WG Types est:

http://www.cs.chalmers.se/ComputingScience/Research/Logic/TypesWG/.

Ces projets sont une collaboration avec environ 15 sites en Europe; notamment les universités de Cambridge et d'Edimbourg, l'équipe Coq de l'INRIA Rocquencourt, et plusieurs logiciens européens, dont Henk Barendregt et Jean-Yves Girard. La communauté *Types* est très active, et réunit de nombreux chercheurs de haut niveau. L'appartenance à cette communauté est un grand atout pour nous. De nombreux chercheurs de cette communauté, ou proches d'elle, travaillent sur les mêmes sujets que nous.

Le dernier contrat se termine en septembre 1999. Il est possible, et même assez probable, que nous tentions de le prolonger sous la même forme *(Working Group)*. En parallèle, nous avons entamé des discussions avec plusieurs projets en vue de monter un projet LTR sur les preuves de correction de logiciel.

Nous détaillons ici les principaux travaux connexes aux nôtres dans cet environnement. Presque tous les sites de Types sont concernés.

8.3.2 Chalmers, Suède

Thierry Coquand à l'université de Chalmers, à Goteborg, recherche comme nous une théorie des types permettant une bonne formalisation des liaisons dans les langages de programmation fonctionnels. Pour l'instant, il recherche une théorie d'ordre un (manipulant directement les variables), mais il est très interessé par nos travaux utilisant la syntaxe abstraite d'ordre supérieur. Thierry Coquand était avec Frank Pfenning l'un des rapporteurs de la thèse de Pierre Leleu [Lel98b].

8.3.3 Helsinski, Finlande

Jan von Plato est l'un des premiers chercheurs a avoir utilisé le système Coq pour réaliser des preuves de correction de protocoles. Il travaille encore dans ce domaine aujourd'hui.

8.3.4 Münich, Allemagne

Le groupe de Tobias Nipkow s'est considérablement investi dans les preuves de propriétés (d'un noyau) du langage Java (hors parallélisme pour l'instant), en Isabelle/HOL, qui font l'objet de 3 thèses de doctorat: une preuve de préservation des types par l'évaluation [NvO98] et une preuve de correction d'un compilateur vers le byte-code.

Ce travail sert aussi d'exemple test au développement de travaux plus généraux sur le système. Ainsi, un environnement dédié à la spécification et aux preuves de propriétés des langages à objets à été implémenté pour Isabelle [NW98].

8.3.5 Cambridge, Grande Bretagne

Larry Paulson réalise des preuves de correction de protocoles en Isabelle et travaille à l'amélioration du système dans le but de faciliter ces preuves depuis plusieurs années.

Gabbay et Andrew Pitts travaillent sur le problème de la représentation des liaisons dans les langages fonctionnels, en suivant une approche qui devrait aboutir à une solution comparable à celle proposée par Andrew Gordon et Tom Melham: des termes définis modulo α-conversion (3.1.2).

8.3.6 Durham, Grande Bretagne

Nous avons vu plus haut que James McKinna et Robert Pollack utilisent pour spécifier des règles de typage (typage des théories typées) une technique d'ordre un intéressante [MP93, MP99, Pol95] qui distingue les variables libres des variables liées. Ils ont ainsi formalisé Lego en Lego (Lego et Coq sont des systèmes très voisins). Ce travail, comprenant l'élaboration et le rafinement de leur méthode, leur a demandé un effort de nombreuses années.

Yu a consacré sa thèse au mariage du model-checking et de la vérification interactive de programmes concurrents en Lego.

8.3.7 Edimbourg, Grande Bretagne

Dans l'équipe de Rod Burstall, Healfdene Goguen et Paul Jackson, en collaboration avec la compagnie Harlequin, ont implémenté une preuve de correction d'un gc en Lego [Jac98].

Thomas Kleymann a consacré sa thèse de doctorat à la preuve de correction de différentes logiques de Hoare utilisées pour la preuve de correction de programmes [Kle98]. Pour illustrer son travail, il a réalisé de nombreux exemples qui devraient nous interesser.

Healfdene Goguen et Adriana Compagnoni ont étudié les propriétés méta-théoriques de systèmes de types pour des langages orientés objets en utilisant la méthode introduite par Healfdene Goguen dans sa thèse *(Typed Operational Semantics)*. Nous avons également utilisé cette technique,

que nous avons du étendre pour l'occasion, dans nos travaux sur la recherche d'une théorie typée permettant la récursion sur des termes de type fonctionnel (3.1.3).

Martin Hofmann a récemment donné la justification [Hof99] des principes d'induction pour les termes d'une syntaxe abstraite d'ordre supérieure que nous avions introduit en 94 et 95 [DH94, DFH95] (section 3.1.2). Cette justification est pour l'instant encore théorique, utilisant les catégories. Elle devrait servir de base à l'extension de notre système à un système généralisant le Calcul des Constructions Inductives.

Enfin, M. Fiore, G. Plotkin et D. Turi proposent une formalisation de la syntaxe abstraite d'ordre supérieur [FPT99] en utilisant les catégories, obtenant par là des modèles qui généralisent l'approche traditionelle.

8.3.8 Londres, Grande Bretagne

Ishtiaq et David Pym ont proposé une théorie des types appelée *Relevant Logical Framework* [IP97], qui correspond à la théorie LF avec un noyau linéaire. Ce système est à comparer à celui que nous avons proposé dans la section 3.1.3, basé sur un noyau modal.

8.3.9 Udine, Italie

Marino Miculan, qui doit venir en séjour post-doctoral chez nous en 99, a consacré sa thèse de doctorat à la formalisation et à l'étude comparative de nombreuses logiques en LF: la Sémantique Opérationelle Structurelle et la Sémantique Naturelle, la logique de Hoare, la logique dynamique introduite par Furio Honsell, des logiques modales et le mu-calcul [Mic97].

Furio Honsell et Marino Miculan ont proposé une description du π-calcul en utilisant la syntaxe abstraite d'ordre supérieure pour formaliser les liaisons du calcul [HMS98]. Notre proposition [DS99] (3.1.1), bien qu'élaborée de manière indépendante, peut être vue comme le prolongement de la leur.

8.3.10 Utrecht, Pays-Bas

Marc Bezem a réalisé plusieurs preuves de correction de protocoles dans le système Coq.

8.4 Collaborations internationales

Les 2 collaborations suivantes se sont déjà concrétisées par de nombreuses visites, l'écriture d'articles en commun et l'échange d'étudiants post-doctoraux; nos directions de recherche sont toujours très voisines. Frank Pfenning maintient une page web très riche sur les *Logical Frameworks*, que le lecteur intéressé pourra consulter avec profit:

http://www.cs.cmu.edu/afs/cs/user/fp/www/lfs.html.

8.4.1 CMU, Penn University, Edimbourg

Nous entretenons une collaboration suivie avec Frank Pfenning (Carnegie Mellon University), et dans une moindre mesure avec Dale Miller (Penn State univ., auparavant à Penn univ.) et l'équipe de Gordon Plotkin (Edimbourg) sur le thème de la syntaxe abstraite d'ordre supérieur et l'induction dans les théories typées. Nous avons entamé des discussions avec ces chercheurs en vue de monter un projet NSF sur la syntaxe abstraite d'ordre supérieur et l'induction dans les théories typées.

8.4.2 ATT Bell Labs

Nous collaborons également avec Amy Felty, de l'équipe de Dave McQueen à ATT Bell Labs sur le thème de la spécification et des preuves de propriétés de langages, depuis de nombreuses années. Amy Felty a passé plusieurs mois dans notre équipe en 91 et Joëlle Despeyroux a passé 2 mois et demi dans l'équipe de Dave McQueen à l'automne 95.

8.4.3 Cornell

Nous restons toujours en contact étroit avec les membres de l'équipe de Robert Constable à Cornell. Le système développé là-bas, NuPrl, ne possède pas de types inductifs. En conséquence, la technique de representation des logiques et langages utilisée en NuPrl est très différente de celle utilisée en Coq, Alf ou Lego: c'est le *shallow embedding*, par opposition au *deep embedding*. La représentation est plus "lache", beaucoup moins intéressante. Cependant, cette équipe comprend de nombreux membres, dont certains ont réalisés de nombreuses preuves, aussi bien de propriétés de langages que de correction de programmes. L'un des exemples les plus récents est la formalisation d'une partie du langages OCaml, d'un protocole de communication appelé Ensemble, écrit en OCaml, et de la preuve de correction de ce protocole.

9 Diffusion des résultats

9.1 Animation de la communauté scientifique

Joëlle Despeyroux est membre nommé de la Commission de Spécialistes à l'Université de Nice Sophia-Antipolis (UNSA) depuis mars 98.

Joëlle Despeyroux est responsable scientifique de l'école d'été internationale Types qu'elle doit organiser, sous l'égide du *Working Group* "Types", à Giens, en septembre 99 (2 semaines). La page web de l'école est:

http://www.inria.fr/croap/personnel/Joelle.Despeyroux/types-sum-school.html.

Joëlle Despeyroux a organisé dans le passé l'un des séminaires techniques du projet *Types*: "On proving properties of programming languages", à l'INRIA Sophia, en septembre 1993.

André Hirschowitz a eu et a toujours de nombreuses responsabilités dans son domaine, la géométrie algébrique. Il a été coordonnateur d'un réseau HCM et de trois euroconférences HCM. Il a initialisé la série des euroconférences europroj (dix dont les trois HCM) des Gael (six) et des Gaf (quatre). Enfin, il est actuellement directeur du GDR GAC. Ces activités, bien que ne concernant pas le thème de notre projet, témoignent de la grande visibilité d'un de ses membres et sont citées pour cette raison.

9.2 Enseignement universitaire

Joëlle Despeyroux est coordinateur local du Dea MDFI (Mathématiques Discrètes et Fondements de l'Informatique) de Marseille depuis octobre 98. Elle y enseigne avec Yves Bertot un module "sémantique naturelle" depuis 1995 (25 heures).

Joëlle Despeyroux a proposé un module optionnel de 25h intitulé "Mécanisation des preuves" au Dea d'informatique de Nice (créé en 95), module dans lequel devrait intervenir Loïc Pottier. Malheureusement, cette option n'a jamais encore été ouverte.

André Hirschowitz est professeur à l'université de Nice (UNSA), dans le département de Mathématiques. Sa spécialité est la géométrie algébrique, discipline dans laquelle il a acquis une grande notoriété et qu'il enseigne depuis de nombreuses années au niveau Dea.

9.3 Thèses

Pierre Leleu a soutenu brillament sa thèse en décembre 98. Il est actuellement chef du Service Études et Grands Travaux à la Direction Départementale de l'Équipement de Seine-Maritime, à Rouen.

Guillaume Gillard est en 2ième année de thèse.

9.4 Stages

Les membres du projet encadrent régulièrement des stagiaires au niveau Dea ou 1ère année ENS. Ces stagiaires viennent le plus souvent des formations parisiennes (Paris 7, CERMICS, X ou ENS). 4 stages sont déjà proposés pour 99.

9.5 Participation à des colloques et séminaires, invitations

Les membres du projet participent régulièrement au séminaire annuel *Types*, dans lequel ils exposent leurs travaux récents ou en cours.

Nous participons le plus régulièrement possible aux conférences LICS et TLCA, et de manière plus récente aux conférences CONCUR et CADE.

Joëlle Despeyroux a été invité à la conférence internationale *on Numerical Analysis and Computer Science with Applications*, qui doit avoir lieu à Plovdiv, en Bulgarie, en aout 99.

André Hirschowitz a été conférencier invité à de nombreuses conférences en géométrie algébrique. Pour la seule année 98, il a été invité à Rome, Bayreuth, Oberwolfach et Berkeley.

9.6 Journaux

Nos travaux peuvent interesser de nombreux journaux. Parmi les plus prestigieux, citons: JAR (Automated Reasoning), JFP (Functional Programming), JIC (Information and Computation), JLC (Logic and Computation), JSC (Symbolic Computation), TCS (Theoretical Computer Science) et JFLP (Functional and Logic Programming).

Nous avons soumis nos travaux en collaboration avec Frank Pfenning à TCS, et une partie de nos travaux en collaboration avec Pierre Leleu à JFLP. Ces deux soumissions ont été acceptées sous réserve de modifications mineures. Nous comptons soumettre une autre partie de nos travaux en collaboration avec Pierre Leleu à JFP.

10 Conclusion!

Notre avis sur notre projet est le suivant: le thème de recherche est cohérent et représente toujours l'un des axes stratégiques de l'INRIA. Il mérite certainement la création d'un projet, d'autant plus que ce projet est déjà bien implanté dans l'INRIA comme au niveau national et international. Notre seule inquiétude est le manque de moyen en hommes au départ. Si l'intérêt scientifique de ce projet est reconnu, ce qui semble déjà être le cas, il devra être aidé rapidement par un apport en capital humain: doctorants, personnel permanent et/ou collaborateurs extérieurs.

Références

[Bar95] B. Barras. Coq en Coq. Mémoire du DEA informatique, mathématiques et applications, École Polytechnique, 1995. INRIA research report RR-3026, October 1996.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997.

[BFM98] Jean Paul Bodeveix, Mamoun Filali, and César Muñoz. Formal support for the B method in Coq and PVS. submitted, October 1998.

[Bol96] Dominique Bolignano. An approach to the formal verification of cryptographic protocols. In *Proceedings of the third ACM Conference on Computer and Communication Security*, March 1996.

[BW97] B. Barras and B. Werner. Coq in Coq. submitted for publication, 1997.

[CDD+85] Dominique Clement, Joëlle Despeyroux, Thierry Despeyroux, Laurent Hascoet, and Gilles Kahn. Natural Semantics on the computer. Research Report RR-416, INRIA, June 1985.

[CDDK86] Dominique Clement, Joëlle Despeyroux, Thierry Despeyroux, and Gilles Kahn. A simple applicative language: Mini-ML. In *Proceedings of the conference Lisp and Functional Programming, LFP'86*. ACM Press, August 1986. Also appears as INRIA Research Report RR-529, May 1986.

[CH96] Pierre Crégut and Barbara Heyd. A modular coding of unity in Coq. In *Proceedings of Theorem Proving in Higher Order Logic, TPHOL'96, Turku, Finland*, August 1996.

[CH97] B. Chetali and Barbara Heyd. Formal verification of concurrent programs in LP and in Coq: A comparative analysis. In *Proceedings of Theorem Proving in Higher Order Logic, TPHOL'97, NJ, USA*, August 1997.

[CT95] Cristina Cornes and Delphine Terrasse. Automating inversion of inductive predicates in Coq. In *Types for Proofs and Programs*, volume 1158 of *Springer-Verlag LNCS*, pages 85–104, June 1995.

[Des86] Joëlle Despeyroux. Proof of translation in Natural Semantics. In *Proceedings of the first Symp. on Logic In Computer Science, LICS'86*. IEEE Computer Society, June 1986. Also appears as INRIA Research Report RR-514, April 1986.

[Des90] Joëlle Despeyroux. Theo: an interactive proof development system. user's manual. Technical Report RT 116, INRIA, February 1990.

[Des92] Joëlle Despeyroux. Theo: an interactive proof development system. *The Scandinavian Journal on Computer Science and Numerical Analysis (BIT), special issue on 'Programming Logic'*, 32:15–29, 1992. A preliminary version appears as INRIA Research Report no. 887, August 1988.

[Des98] Joëlle Despeyroux. Sémantique Naturelle: Spécifications et Preuves. INRIA Research Report RR-3359, February 1998. Lecture notes for a graduate course in the "DEA" “Mathématiques Discrètes et Fondements de l'Informatique” (MDFI), Marseille, 1995-1999 (80 pages, in french).

[DFH95] Joëlle Despeyroux, Amy Felty, and André Hirschowitz. Higher-order abstract syntax in Coq. In M. Dezani and G. Plotkin, editors, *Proceedings of the TLCA 95 Int. Conference on Typed Lambda Calculi and Applications*, volume 902, pages 124–138. Springer-Verlag LNCS, April 1995. Also appears as INRIA Research Report RR-2556.

[DG94] Damien Doligez and Georges Gonthier. Portable, unobtrusive garbage collection for multiprocessor systems. In *Proceedings of the POPL ACM Conference on Principles of Programming Languages*, pages 70–83, January 1994.

[DH94] Joëlle Despeyroux and André Hirschowitz. Higher-order syntax and induction in Coq. In F. Pfenning, editor, *Proceedings of the fifth Int. Conf. on Logic Programming and Automated Reasoning (LPAR 94)*, volume 822, pages 159–173. Springer-Verlag LNAI, July 1994. Also appears as INRIA Research Report RR-2292 (June 1994).

[DL98] Joëlle Despeyroux and Pierre Leleu. A modal λ-calcul with iteration and case constructs. In *Proc. of the annual Types for Proofs and Programs seminar*, Springer-Verlag LNCS, March 1998.

[DL99a] Joëlle Despeyroux and Pierre Leleu. Metatheoretic results for a modal λ-calcul. Journal version. Submitted for publication in 98-99, 1999.

[DL99b] Joëlle Despeyroux and Pierre Leleu. Primitive recursion for higher-order abstract syntax - dependant types. Draft, submitted for publication, 1999.

[DL99c] Joëlle Despeyroux and Pierre Leleu. Towards primitive recursion over objects of functional type. Journal version. Draft, 1999.

[DPS96] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. Technical Report CMU-CS-96-172, Carnegie Mellon University, Pittsburgh, Pennsylvania, August 1996.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In Philippe de Groote and J. Roger Hindley, editors, *Proceedings of the TLCA 97 Int. Conference on Typed Lambda Calculi and Applications, Nancy, France, April 2–4*, pages 147–163. Springer-Verlag LNCS 1210, April 1997. An extended version is available as CMU Technical Report CMU-CS-96-172.

[DS99] Joëlle Despeyroux and Davide Sangiorgi. A higher-order specification of the pi-calculus. Submitted for publication, March 1999.

[Fil97] J.-C. Filliâtre. Proof of Imperative Programs in Type Theory. Research Report 97–24, LIP - ENS Lyon, July 1997.

[FPT99] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. draft, January 1999.

[Gil97] Guillaume Gillard. Mini-ML: spécifications en sémantique Naturelle et preuves dans Coq. Rapport de stage de DEA, Paris 7 University, September 1997.

[Gim96] E. Giménez. *Un Calcul de Constructions Infinies et son application à la vérification de systèmes communicants*. Thèse d'université, Ecole Normale Supérieure de Lyon, December 1996.

[GM96] Andrew Gordon and Tom Melham. Five axioms of alpha-conversion. In *Proceedings of the int. conf. on Theorem Proving in Higher Order Logics, Turku, Finland, August 1996*, Springer-Verlag LNCS?, 1996.

[Hen98] Loïc Henry_Greard. A proof of type preservation for the pi-calculus in Coq. Research Report RR-XXX, Inria, December 1998. To appear. Available in the Coq Contrib library.

[Hey97] Barbara Heyd. *Application de la théorie des types et de l'assistant Coq à la vérification de programmes parallèles*. Phd thesis, Nancy University, France, December 1997. In French.

[Hir97] Daniel Hirschkoff. A full formalization of pi-calculus theory in the Calculus of Constructions. In Elsa Gunter, editor, *Proceedings of the International Conference on Theorem Proving in Higher Order Logics*, Murray Hill, New Jersey, August 1997.

[Hir99] Daniel Hirschkoff. *Mise en oeuvre de preuves de bisimulation*. Phd thesis, École Nationale des Ponts et Chaussées (ENPC), January 1999. In French.

[HKPM97] G. Huet, G. Kahn, and Ch. Paulin-Mohring. The Coq proof assistant - a tutorial, version 6.1. Technical Report 204, INRIA, Août 1997. Version révisée distribuée avec Coq.

[HMS98] Furio Honsell, Marino Miculan, and Ivan Scagnetto. Pi-calculus in (co)Inductive Type Theory. submitted, 1998.

[Hof99] Martin Hofmann. Semantical analysis of higher-order abstract syntax. Draft, submitted for publication, 1999.

[Hue94] G. Huet. Residual theory in λ-calculus: A complete gallina development. *Journal of Functional Programming*, July 1994. Voume 4, part 3, pages 371–394.

[IP97] S. Ishtiaq and D. Pym. A relevant analysis of natural deduction. Draft, submitted for publication, 1997.

[Jac98] Paul Jackson. Verifying a garbage collection algorithm. In *Proceedings of Theorem Proving in Higher Order Logic, TPHOL'98*, 1998.

[Kle98] Thomas Kleymann. *Hoare Logic and VDM: Machine-checked Soundness and Completeness Proofs*. PhD thesis, Edinburgh University, 1998.

[KT96] Delphine Kaplan-Terrasse. Vers la certification du compilateur v5 d'Esterel. Rapport de Post-Doc, Inria-Sophia, France, a paraître en rapport Inria, janvier 99, September 1996.

[Lau97] Olivier Laurent. Sémantique Naturelle et Coq: vers la spécification et les preuves sur des langages à objets. Research Report RR-3307, INRIA, November 1997.

[Lel97] Pierre Leleu. A modal λ-calcul with iteration and case constructs. Research Report RR-3322, INRIA, December 1997.

[Lel98a] Pierre Leleu. Metatheoretic results for a modal λ-calcul. Research Report RR-3361, INRIA, February 1998.

[Lel98b] Pierre Leleu. *Syntaxe abstraite d'ordre supérieur et récursion dans les théories typées*. Phd thesis, École Nationale des Ponts et Chaussées (ENPC), December 1998. In French.

[LP92] Zhaohui Luo and Robert Pollack. The LEGO proof development system: A user's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh, May 1992.

[Mic97] Marino Miculan. *Encoding Logical Theories of Programs*. PhD thesis, University of Pisa, February 1997.

[ML80] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North-Holland, 1980.

[ML85] Per Martin-Löf. Truth of a propositions, evidence of a judgement, validity of a proof. Notes to a talk given at the workshop *Theory of Meaning*, Centro Fiorentino di Storia e Filosofia della Scienza, June 1985.

[MN94] Lena Magnusson and Bengt Nordström. The ALF proof editor and its proof engine. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs*, pages 213–237. Springer-Verlag LNCS 806, 1994.

[MP93] James McKinna and Robert Pollack. Pure Type Sytems formalized. In M.Bezem and J.F.Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 289–305. Springer-Verlag LNCS 664, March 1993.

[MP99] James McKinna and Robert Pollack. Some Pure Type Sytems formalized. *To appear in a special issue on Formal Proof of the Journal of Automated Reasoning (JAR)*, 1999.

[Nor93] Bengt Nordström. The ALF proof editor. In *Proceedings of the Workshop on Types for Proofs and Programs*, pages 253–266, Nijmegen, 1993.

[NvO98] Tobias Nipkow and David von Oheimb. Java light is type-safe – definitely. In *Proceedings of the POPL ACM Conference on Principles of Programming Languages*, pages 161–170, 1998.

[NW98] Wolfgang Naraschewski and Markus Wendel. Object-oriented verification based on record subtyping in higher-order logic. In *Proceedings of Theorem Proving in Higher Order Logic, TPHOL'98*, pages 349–366, 1998.

[Par95a] C. Parent. Synthesizing proofs from programs in the calculus of inductive constructions. In *Third International Conference on the Mathematics of Program Construction*, number 947 in Lecture Notes in Computer Science. Springer-Verlag, Juillet 1995.

[Par95b] C. Parent. *Synthèse de preuves de programmes dans le Calcul des Constructions Inductives.* Thèse d'université, École Normale Supérieure de Lyon, January 1995.

[Par97] C. Parent. Verifying programs in the calculus of inductive constructions. *Formal Aspects of Computing*, 1997. A paraître.

[PM92] Christine Paulin-Mohring. Inductive definitions in the system coq. rules and properties. In J.F. Groote M. Bezem, editor, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*, Springer-Verlag LNCS 664, pages 328–345, 1992. also available as a Research Report RR-92-49, Dec. 1992, ENS Lyon, France.

[PM96] C. Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur.* Habilitation à diriger les recherches, Université Claude Bernard Lyon I, December 1996.

[Pol94] Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions.* PhD thesis, University of Edinburgh, 1994.

[Pol95] Robert Pollack. A verified typechecker. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 365–380, Edinburgh, Scotland, April 1995. Springer-Verlag LNCS 902.

[Ter95a] Delphine Terrasse. Encoding Natural Semantics in Coq. In *Proceedings of the Fourth International Conference on Algebraic Methodology and Software Technology, AMAST'95*, volume 936 of *Springer-Verlag LNCS*, pages 230–244, July 1995.

[Ter95b] Delphine Terrasse. *Vers un environnement de développement de preuves en Sémantique Naturelle.* Thèse de doctorat, École Nationale des Ponts et Chaussées (ENPC), octobre 1995. In French.

[Wer94] B. Werner. *Une Théorie des Constructions Inductives.* PhD thesis, Université Paris VII, Mai. 1994.

Table des matières